**********

ATM Forum Document Number: ATM Forum/95-0178R1

**********

Title: A Sample Switch Algorithm

**********

Abstract:
Describes the switch algorithm developed at OSU. The algorithm is fully compatible with the latest source and switch requirements and has been thoroughly tested. The text to be included in the TM document appendix is presented.

**********

Source:

Raj Jain, Shiv Kalyanaraman, Ram Viswanathan, and Rohit Goyal
The Ohio State University

Raj Jain is now at Washington University in Saint Louis, jain@cse.wustl.edu http://www.cse.wustl.edu/~jain/

The presentation of this contribution at the ATM Forum is sponsored by NIST.

**********

Date: February 6-10, 1995

**********

Distribution: ATM Forum Technical Working Group Members
(Traffic Management)

**********

Notice: This contribution has been prepared to assist the ATM Forum. It is offered to the Forum as a basis for discussion and is not a binding proposal on the part of any of the contributing organizations. The statements are subject to change in form and content after further study. Specifically, the contributors reserve the right to add to, amend or modify the statements contained herein.

**********

It is proposed that the following text be added as an appendix to the TM document.

Appendix: A Sample Switch Algorithm

The traffic management requirements for switches allow considerable flexibility to vendors in deciding what feedback to give to the sources. This appendix describes a sample switch feedback algorithm that is compatible with the current source, switch, and destination requirements. It is designed for explicit rate switches. It has been tested thoroughly in simulations. It is included here only as an example.

Design Goals: The switch algorithm is named Explicit Rate Indication for Congestion Avoidance (ERICA) algorithm. The design goals of this algorithm were to provide the following features:

1. Congestion Avoidance: The algorithm tries to keep cell queue lengths small while keeping the link utilization high. Small queue lengths results in small cell transfer delay (CTD) and small cell transfer delay variation (CTDV).

2. Max-Min Optimality: The ERICA algorithm converges to the max-min optimal. Variations of this algorithm for other optimality criteria can be easily developed.

3. Fast Response: Small queue lengths result in fast feedback to the source. Under overload, the rates come down fast. During underload, the rates go up fast, thereby, optimizing the transient performance.

4. Small Number of Parameters: Switches implementing ERICA have only two parameters: target utilization and load averaging interval. The utilization parameter should be set close to 95%. A higher value would result in higher queue length under overload. The load averaging interval is specified in number of cells. For 155 Mbps links, the default value is 30 cells.

5. Insensitivity to Parameters: If the user sets the parameter incorrectly, the performance degrades gracefully. For example, if the target link utilization is set to 80%, the link still operates but the maximum utilization under steady state is 80%. Of course, during overloads the link may be fully utilized.

6. Easy to set parameters: Setting the switch parameters is easy. Most network system managers are familiar with the term link utilization and can set the parameter correctly.

7. High-Start Feasible: One side benefit of fast response is that the rates (and hence the queue lengths) decrease rapidly during overload. The sources can start at any rate including the peak cell rate. The queues may build up but are reduced quickly along with the allowed cell rates. Although, high-start may not be recommended under many circumstances, the scheme works even if the VCs start at PCR. Starting at PCR may be desirable in the LAN environment.

OVERVIEW OF THE SCHEME:

The scheme consists of switches monitoring the load on each outgoing link and determining the overload factor and the number of active VCs. The switch measures the time until the Nth cell arrives. If this time is T seconds, the input rate is N/T cells per second. If the rated capacity of the link is C cells per second and the desired target utilization is U, the overload factor is computed as follows:

Input Rate = N/T cps
Target Cell Rate = U*C cps
Overload Factor = Input Rate/Target Cell Rate = N/(T*U*C)

During the next measurement interval of N cells, the switch may simply ask all VCs to change (increase or decrease) their load by the overload factor. Based on loading considerations only, the explicit rate for a VC with current cell rate of CCR is given by:

ER based on load = ER1 = CCR/Overload Factor

Fairness is achieved by ensuring that every VC gets at least the Fair_Share, which is computed as follows:

Fair Share = Target Cell Rate/Number of Active VCs

The number of active VCs is the number of distinct VCs that were seen during the last measurement interval (of N cells).

Combining the fair share with the explicit rate based on load we get:

ER based on load and fairness = ER2 = Max{Fair Share, ER based on load}

If this computed value is lower than the explicit rate field in the RM cell, the switch replaces the explicit rate field in the cell with the computed value.

ER in Cell = Min{ER in Cell, ER based on load and fairness}

There are a few fine details that should be pointed out:

1. The input rate, number of active VCs, and overload is measured in the forward direction while the feedback is sent in RM cells going in the reverse direction. The CCR value in the RM cells going in the forward direction is more current and has a direct relationship with the overload. The CCR value in RM cells returning in the reverse direction may be out of date. It is therefore important to use the latest value of CCR in computing ER1. The switches note this value in the VC table for all forward-going RM cells and use it when computing explicit rate indication for reverse-going RM cells.

2. The switches measure load over one measurement interval of N cells and use it over the next measurement interval of N cells. If several reverse-going RM cells from a VC are seen during one interval, they should all carry the same explicit rate indication to the VC. The switches, therefore, store ER2 in the VC table and also keep an indication that a reverse-RM cell has been seen for the VC in that measurement interval. Not following this rule results in oscillatory behaviour due to confusing feedback given to the source.

3. In computing the explicit rate based on load and fairness, ERICA algorithm allows the *maximum* of the Fair_Share and the explicit rate based on load alone. This prevents VCs from being unnecessarily damped and allows fast response time. Not following this rule may result in low rate VCs not being able to come up.

4. The switch generated BECN option can be optionally used during the first round-trip on new VCs. This is particularly helpful in WAN configurations where the round-trip feedback delay may be large. Although the standard allows BECNs after the first round-trip, there is little advantage in using it.

PSEUDO CODE:

1. Parameters: For each output port (or link), the system manager sets the the following parameters:

```
int Measurement_Interval_In_Cells; /* Measurement Interval in Cells */
real Target_Utilization; /* Target Utilization between 0 and 1 */
real Link_Cell_Rate; /* Link capacity in cells per second */
```

The default suggested values of the first two parameters are 30

cells and 0.95, respectively. For a 155 Mbps link, the link cell rate is 155,000,000/(8*53) or 365,566 cells per second.

2. Data Structure: The switch maintains the following variables:

A. For each output port (or link):

```
int In_Count; /* Number of cells received for that link */
real Target_Cell_Rate; /* Target cell rate in cps */
int Number_Of_VCs; /* Total number of VCs on this link */
int Number_Of_Active_VCs; /* VCs seen in the last interval */
real Measurement_Interval_Start_Time;
```

B. For each VC:

```
boolean Cells_Seen; /* One or more cells were seen
                       in this measurement interval*/
real Last_Seen_CCR; /*  CCR last seen in the forward direction */
boolean Reverse_RM_Cells_Seen_In_This_Interval;
real Last_ER; /* Last explicit rate feedback given for this VC */
```

3. Initialization:

```
/* For all ports, initialize the number of cells received */
for(i=0; i<Number_Of_Ports; i++){
    Output_Port[i].In_Count = 0;

/* Length of switch measurement interval in cells */
    Initialize Output_Port[i].Measurement_Interval_In_Cells;
                      /* Eg: 30 cells */
                   /* But may depend on link bandwidth */

    Initialize Output_Port[i].Number_Of_Active_VCs;
        /* Initialize the number of active VCs at each port */

    Output_Port[i].Target_Cell_Rate
                = Target_Utilization * Link_Cell_Rate;

    Output_Port[i].Fair_Share = (Output_Port[i].Target_Cell_Rate/
                            Output_Port[i].Number_Of_Active_VCs );

    /* Reset Cells_Seen */
    For(j=0; j < Number_Of_VCs; j++)
        {
        Output_Port[i].Reverse_RM_Cells_Seen_In_This_Interval[j] = 0;

/* Cells_Seen is used to count the number of active VCs */
    For(j=0; j <  Number_Of_VCs; j++)
        Output_Port[i].Cells_Seen[j] = 0;

/* measuring a switch interval */
    Output_Port[i].Measurement_Interval_Start_Time = now;

}
/* ------------------ normal run ----------------------------*/

if(Cell_Received(i,j))
{

     /* This means that cell for  vc j has been received at port i */

    Output_Port[i].In_Count++;

    /* For counting the number of active VCs */
    Output_Port[i].Cells_Seen[j] = 1;
```

```
    If (Output_Port[i].In_Count >= Output_Port[i].
                Measurement_Interval_In_Cells)
        End_Of_Interval_Calculations(i);

}

/* ------------------ RM cell processing ----------------------------*/

     if( RM cell is received){

      if( cell->DIR == forward) {    /* forward path */

         /* i th output port */
        /* Note the latest rate seen on the forward path */
         Output_Port[i].Last_Seen_CCR[cell->VC_Number] = cell->CCR;

       /* Include the BECN option here if it is being implemented */

      }

      else if(cell->DIR == backward) { /* reverse path */

      /* Port corresponding to the forward link */
      i = Forward_Link_Port;

      /* New feedback only if an RM  cell of this VC has not yet been
         seen in this interval */

         if( Output_Port[i].
             Reverse_RM_Cell_Seen_In_This_Interval[cell->VC_Number] != 1)
         {

         /* core algorithm */

         ER1 = (Output_Port[i].Last_Seen_CCR[cell->VC_Number]/
              Output_Port[i].Overload);
         ER2 = Max(ER1, Output_Port[i].Fair_Share);
         New_ER = Min( Output_Port[i].Target_Cell_Rate, ER2);

         if(New_ER < cell->ER)
             cell->ER = New_ER;

        /* Switches compute only one feedback per measurement interval */
        /* Same feedback is given if other RM cells follow */

         Output_Port[i].
             Reverse_RM_Cells_Seen_In_This_interval[cell->VC_Number] = 1;

         /* Switches dont inhibit other switches' feedback if
                They have shorter intervals */
         /* They simply advertise one rate per VC throughout
                The interval */
         Output_Port[i].Last_ER[cell->VC_Number] = New_ER;

         }  /* end:First reverse RM cell in this interval */

         else{
          /* More than one reverse RM cell seen in this interval */
              /* Give out the same ER for all RMs of this VC */
             New_ER =  Output_Port[i].Last_ER[cell->VC_Number] ;
             if(New_ER < cell->ER)
                     cell->ER = New_ER;
              } /* end : Second RM cell seen in the same interval */
```

```
     }  /* end : reverse direction */
          New_VC = 0;
} /* end : RM  cell */


/* ------------- end of interval processing ------------------*/

End_Of_Interval_Calculations(int i)
{

    Interval_Length = now -
                      Output_Port[i].Measurement_Interval_Start_Time;
    Output_Port[i].Measurement_Interval_Start_Time = now;

    /* overload computation */
    Output_Port[i].Overload  = ( Output_Port[i].In_Count /
                  (Output_Port[i].Target_Cell_Rate*Interval_Length) );

    Output_Port[i].Number_Of_Active_VCs = Get_Number_Of_Active_VCs(i);

    Output_Port[i].Fair_Share = ( Output_Port[i].Target_Cell_Rate/
                               Output_Port[i].Number_Of_Active_VCs );

     Output_Port[i].In_Count = 0;

    /* Init RM cells */
    For(j=0; j <  Output_Port[i].Number_Of_VCs; j++)
         Output_Port[i].Reverse_RM_Cells_Seen_In_This_IntervalCell[j]=0;

}
Get_Number_Of_Active_VCs(int i)
{
    int count = 0;
    int j;

    for ( j =0; j< Output_Port[i].Number_Of_VCs; j++){

      if(Output_Port[i].Cells_Seen[j] ==1)
        count = count + 1;
      Output_Port[i].Cells_Seen[j] = 0;

    }
}
```